CryptoCAN Datasheet

Key features

Supports publish/subscrbe model

CAN is designed as a publish/subscribe fieldbus: a single sender of a CAN frame publishes data in a CAN frame (typically containing sensor data) and multiple subscribers to the frame unpack the data they want. CryptoCAN directly supports this model: there is no peer-to-peer communication.

Standard cryptographic functions

CryptoCAN does not use proprietary cryptographic functions and relies on industry standard encryption algorithms: AES-128 for encryption, AES-CMAC for message authentication, CFB mode, and AUTOSAR SHE functions for random number generation and key distribution.

Standard AUTOSTAR SHE HSM support

CryptoCAN is designed to work with the AUTOSAR Secure Hardware Extensions standard for Hardware Security Modules: this HSM is widely available on microcontrollers aimed at the automotive industry.

Tiny resource usage

CryptoCAN is targeted as resource constrained microcontrollers. The pure software CryptoCAN stack takes less than 3Kbytes of code space and an AES-128 encrypt operation takes just 15 microseconds on an Arm Cortex M0 at 133MHz.

Fast start up

In mission critical systems it is essential that a system can join a running system in a very short time. CryptoCAN was designed to allow receivers to join communication at any time without needing to negotiate sessions with a sender.

Bounded latencies for messaging

CryptoCAN creates two CAN frames with the same priority and timing as the plaintext CAN frame. The latency of the second CAN frame is the latency of the ciphertext message and its latency meets the requirements of existing CAN timing analysis to calculate worst-case latencies.

Debug support

CryptoCAN has special support for debugging where the encryption wrapper is disabled (but preserving the authentication code) where the original frame plaintext is visible to existing CAN tools. The CPU time with this debug option enabled is identical (down to the clock cycle), allowing testing to take place without encryption and then it enabled in the final step without invalidating real-time testing.

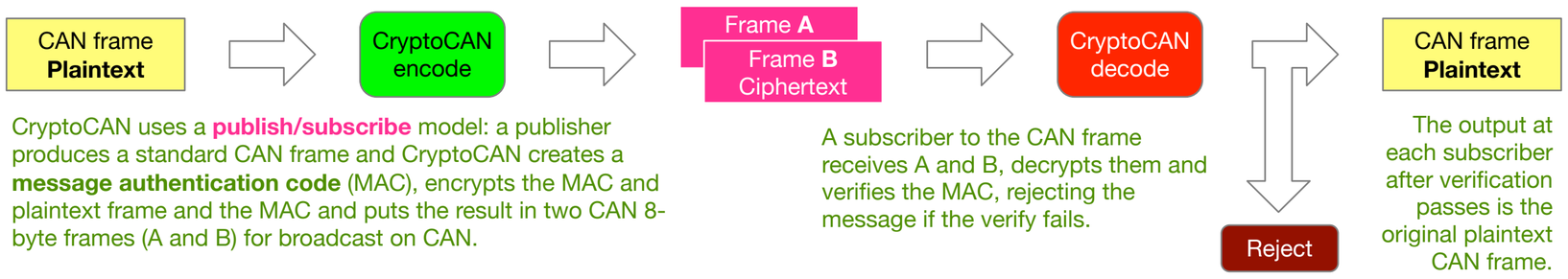
Pure software option

CryptoCAN is designed to work with a standard HSM but many devices do not have this hardware. A software emulated HSM is available to provide a pure software solution. This is ideal for retrofitting encryption into existing CAN systems.

Protection against replay attacks

CryptoCAN has support for preventing replay attacks by including a 31-bit freshness value in computing the message authentication code. Special support is included to resolve synchronisation of freshness values between the time generated at the sender and the time the frames arrive at a receiver.

The publish/subscribe model



CryptoCAN uses a **publish/subscribe** model: a publisher produces a standard CAN frame and CryptoCAN creates a **message authentication code** (MAC), encrypts the MAC and plaintext frame and the MAC and puts the result in two CAN 8-byte frames (A and B) for broadcast on CAN.

A subscriber to the CAN frame receives A and B, decrypts them and verifies the MAC, rejecting the message if the verify fails.

The output at each subscriber after verification passes is the original plaintext CAN frame.

CryptoCAN stack options

On-chip SHE HSM

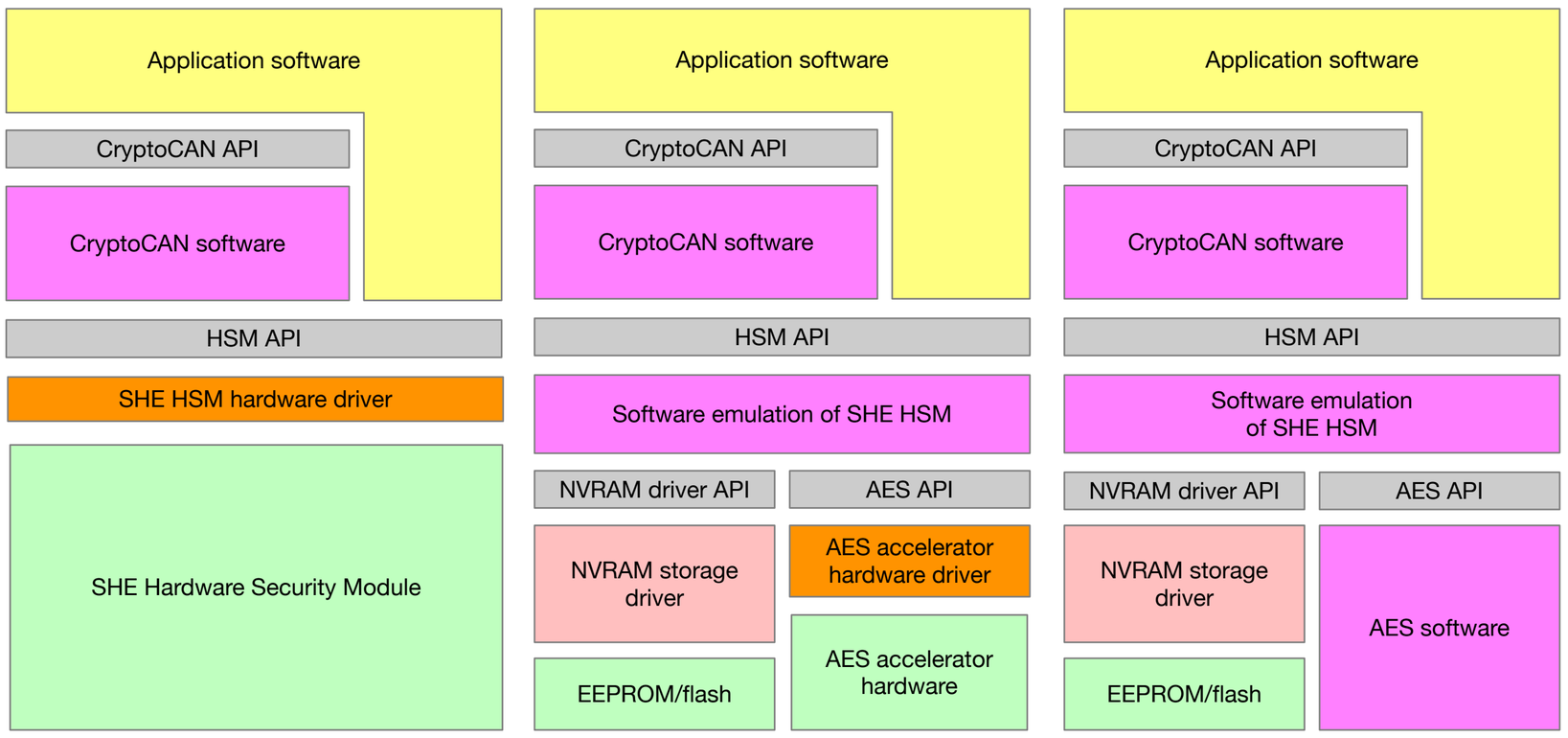
CryptoCAN runs on top of a hardware security module (HSM) API that is a subset of the SHE HSM. For hardware with an on-chip SHE HSM then only the CryptoCAN stack is needed.

AES-128 accelerator hardware

For devices with on-chip AES-128 accelerators CryptoCAN can run on top of an emulated SHE HSM that uses the AES accelerator hardware to speed up the cryptographic operations. Call-backs are defined for storing keys in EEPROM or flash memory.

Pure software implementation

For devices with no cryptographic hardware an emulated SHE HSM using a pure software implementation of AES-128 is provided. Call-backs are defined for storing keys in EEPROM or flash memory.



CryptoCAN software

Execution times

Execution times are given for the pure software emulated SHE HSM (for these figures the software HSM was built with key cache). Measurements were taken on an RP2040 microcontroller (Cortex M0 clocked at 133MHz). Code was located in RAM to avoid eXecute-In-Place (XIP) cache memory delays.

Function	SHE HSM operations	With anti-replay	Without anti-replay
Create frames	1 x `ENC_ECB` + 1 x `GENERATE_MAC`	31.222µs	30.158µs
Receive frame A	1 x `ENC_ECB`	14.310µs	14.310µs
Receive frame B	1 x `VERIFY_MAC`		
Verify failed		18.270µs	18.078µs
Verify OK, replay fail		18.294µs	
Received OK		18.526µs [1]18.486µs	18.206µs

Note [1] Receive OK first time

Memory usage

The CryptoCAN software layer was compiled with gcc for the Arm Cortex M0. The CryptoCAN layer totals of **less than 1Kbyte** of program code. RAM usage is per context: one context is required for a given CAN frame source (e.g. for J1939 one context per PGN used).

Code size

Base	44 bytes
Transmit	380 bytes
Receive	560 bytes

RAM data

Per context	
Transmit	24 bytes
Receive	40 bytes

Software emulation of SHE HSM

Execution times

Execution times for pure software emulated SHE HSM are given for two build variants: a key cache variant and a no-key-cache variant, allowing a CPU time vs. RAM tradeoff. Measurements were taken on an RP2040 microcontroller (Cortex M0 clocked at 133MHz). Code was located in RAM to avoid eXecute-In-Place (XIP) cache memory delays and variability in execution time: cryptographic operations have fixed execution times when the CPU uses no cache or prefetch.

Function	SHE command	With key cache		No key cache	
Random number	`RND`	13.654µs		13.654µs	
AES-128 encrypt	`ENC_ECB`	13.662µs		19.214µs	
AES-128 decrypt	`DEC_ECB`	42.430µs		47.822µs	
CMAC generate	`GENERATE_MAC`	[1]14.552µs	[2]28.152µs	[1]20.086µs	[2]33.686µs
CMAC verify	`VERIFY_MAC`	[1]15.470µs	[2]29.070µs	[1]20.998µs	[2]34.598µs

Note [1] 128-bit MAC message
[2] 256-bit MAC message

Memory usage

The HSM emulation software was compiled with gcc for the Arm Cortex M0. CryptoCAN uses only a subset of the functions: a total of **less than 2Kbytes**. The ROM data was located in RAM on the RP2040 to ensure cryptographic operations have fixed execution times. The RAM for the key cache is used only with the key cache variant and can be eliminated if the no-key-cache build variant is used.

Code size

AES-128 encrypt	860 bytes
AES-128 decrypt	1076 bytes
CMAC	628 bytes
Random number	308 bytes
Extend seed	212 bytes
Export key	456 bytes
Load key	1100 bytes

ROM data

AES-128 encrypt	1024 bytes
AES-128 decrypt	1280 bytes
Constants	112 bytes

RAM data

Key cache	3072 bytes
PRNG data	193 bytes

Note Used by CryptoCAN

Development

C SDK

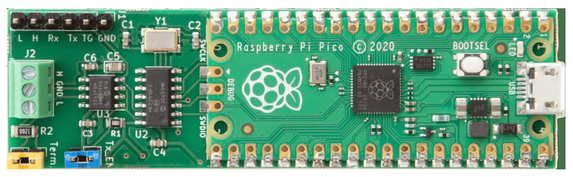
CryptoCAN is distributed as generic C source with a software emulated SHE HSM. There are several build options:

`SM_RAM_TABLES`	*Put ROM tables into RAM*
`SM_CPU_LITTLE_ENDIAN`	*CPU is little-endian*
`SM_KEY_EXPANSION_CACHED`	*Enable key cache for faster AES-128*
`SM_CODE_IN_RAM`	*Put software HSM functions in RAM*
`CC_CODE_IN_RAM`	*Put CryptoCAN functions in RAM*

CryptoCAN is independent of any specific CAN inplementation: it operates on an abstract CAN frame type and the application converts between abstract frames and the specific CAN driver or hardware in the system.

MicroPython SDK

Canis Labs has built custom MicroPython firmware for the CANPico board. It uses the CryptoCAN C SDK to provide a CryptoCAN to the existing Python CAN API and a Python API to an HSM class, with key storage in on-board flash memory (including flash wearing). This MicroPython SDK is intended as an explorer and rapid prototyping development kit for CryptoCAN. More on the CANPico hardware go to canislabs.com/canpico.



v220824-1